Simulation of a 2-link Brachiating Robot with Open-Loop Controllers

David Ufford
Northwestern University
June 2009

1. Project Overview

The goal of this project was to write a complete simulation of a 2-link swinging robot (the "MonkeyBot"). The simulation models swinging, free flight, and impacts with user-inputted parameters.

A secondary goal of this project was to attempt to identify the stability of the MonkeyBot with an open-loop controller. Gaits for horizontal swinging and their dependence on initial conditions were examined.

2. MonkeyBot Overview

The MonkeyBot is a two-link robot, currently in the prototype stage, which is designed to climb along vertical walls. It attaches to the wall surface using electromagnets at the end of each link. These magnets are configured to allow the device to pivot about the attachment point. The electromagnets can be simultaneously or individually activated as needed.

The two links are connected by an actuated rotating joint. With proper control, the motor at this joint can leverage the 2-link swinging dynamics to create desired motion. A combination of swinging the links and changing the activated electromagnet will allow the robot to move about the wall.

The MonkeyBot is modeled as two massive links connected by a central joint. The links of length L_1, L_2 each have mass m_1, m_2 and rotational inertia (I_1, I_2). The center of gravity (CG) is located axially a distance r_1, r_2 along the link.

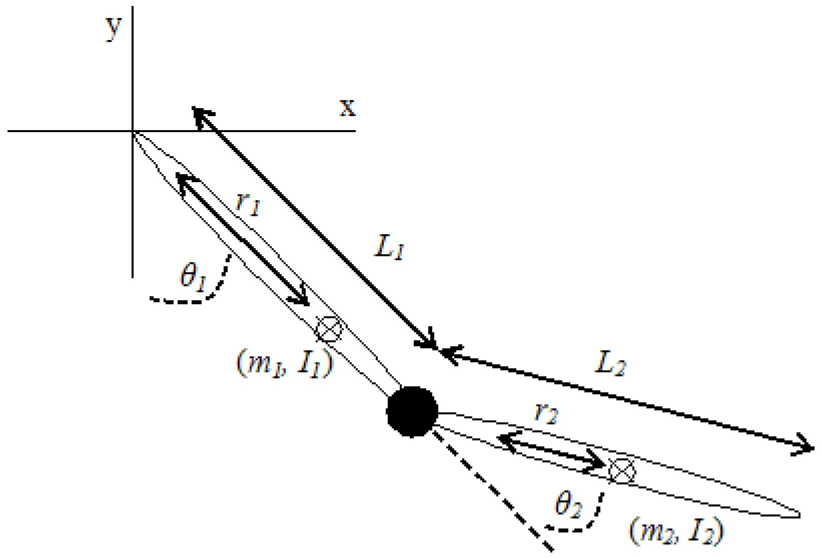

Figure 1: Diagram of 2-link MonkeyBot model with parameters.

If we contain the device to planar movement, the system can be fully expressed in terms of four generalized coordinates, contained in vector *q*.

$$q = \begin{bmatrix} x \\ y \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

(x, y) is the coordinate location of the "top" link. θ_1 is the angle of the top link relative to vertical, and θ_2 is the relative angle between the links.

The MonkeyBot is capable of engage in three general types of motion phases that are defined by the configuration of active magnets

1. Free flight - no magnets are active, system has four degrees of freedom.
2. Swinging – one magnet is activated, system has two degrees of freedom.
3. Fixed – both magnets activated so that the robot cannot move. Dynamics during this state are trivially zero, and are not analyzed.

Transitioning to the swinging or fixed phase involves activating the magnet on one or both links. The resultant clamping of the magnet to the wall surface is a collision that must be accounted for in dynamics and motion planning.

3. MonkeyBot Dynamics

3.1 Motion Dynamics

The simplest way to obtain the mechanical dynamics of our system is via Lagrangian dynamics. The Lagrangian L of the system is defined as the difference between the kinetic *(K)* and potential *(V)* energies of the system.

$$L(q,\dot{q}) = K(q,\dot{q}) - V(q)$$

For our system, the KE and PE terms can be expressed as:

$$K = \frac{1}{2}\left[m_1 v_1^2 + m_2 v_2^2 + I_1 \dot{\theta}_1^2 + I_2 (\dot{\theta}_1 + \dot{\theta}_2)^2 \right]$$

$$V = g(m_1 y_1 + m_2 y_2)$$

v_1 and v_2 are velocities of the CGs expressed in the x-y coordinate system. Likewise y_1 and y_2 are the vertical positions of the CG in the x-y system. They are used in the previous equations for simplicity and conciseness, and are replaced by variables in the *q* coordinate system using the following transformations:

$$y_1 = y - r_1 \sin(\theta_1)$$

$$y_2 = y - L_1 \sin(\theta_1) - r_2 \sin(\theta_1 + \theta_2)$$

$$v_1^2 = \left[\dot{x} + r_1 \cos(\theta_1)\dot{\theta}_1 \right]^2 + \left[\dot{y} + r_1 \sin(\theta_1)\dot{\theta}_1 \right]^2$$

$$v_2^2 = \left[\dot{x} + L_1 \cos(\theta_1)\dot{\theta}_1 + r_2 \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \right]^2 + \left[\dot{y} + L_1 \sin(\theta_1)\dot{\theta}_1 + r_2 \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \right]^2$$

Once the substitutions are made in K and V for the generalized *q* coordinates, we can generate the Euler-Lagrange equations of motion, written below. *F* is the vector of applied external forces in the generalized *q* system.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = F$$

The Euler-Lagrange gives us a system of 4 differential equations, one for each coordinate in *q*. The equations can then be manipulated and grouped into the standardized form:

$$F = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) \qquad \textbf{*(1)}$$

Where *M(q)* is the mass or inertia matrix, *C* is a matrix that contains centrifugal and coriolis terms, and *g* is a vector of gravitational forces. For our 2-link system, these matrices are:[i]

$$M(q)=\begin{bmatrix} m_1+m_2 & 0 & (m_1r_1+m_2L_1)c_1+m_2r_2c_{12} & m_2r_2c_{12} \\ 0 & m_1+m_2 & (m_1r_1+m_2L_1)s_1+m_2r_2s_{12} & m_2r_2s_{12} \\ (m_1r_1+m_2L_1)c_1+m_2r_2c_{12} & (m_1r_1+m_2L_1)s_1+m_2r_2s_{12} & I_1+I_2+m_1r_1^2+m_2L_1^2+m_2r_2^2+2m_2r_2L_1c_2 & m_2r_2^2+m_2r_2L_1c_2+I_2 \\ m_2r_2c_{12} & m_2r_2s_{12} & m_2r_2^2+m_2r_2L_1c_2+I_2 & I_2+m_2r_2^2 \end{bmatrix}$$

$$C(q,\dot{q}) =$$

$$\begin{pmatrix} 0 & 0 & -[(m_1r_1+m_2l_1)s_1\dot{\theta}_1+(m_2r_2s_{12})(\dot{\theta}_1+\dot{\theta}_2)] & -m_2r_2s_{12}(\dot{\theta}_1+\dot{\theta}_2) \\ 0 & 0 & (m_1r_1+m_2l_1)c_1\dot{\theta}_1+m_2r_2c_{12}(\dot{\theta}_1+\dot{\theta}_2) & m_2r_2c_{12}(\dot{\theta}_1+\dot{\theta}_2) \\ 0 & 0 & -m_2r_2l_1s_2\dot{\theta}_2 & -m_2r_2l_1s_2(\dot{\theta}_1+\dot{\theta}_2) \\ 0 & 0 & m_2r_2l_1s_2\dot{\theta}_1 & 0 \end{pmatrix}$$

$$G(q) =$$

$$\begin{pmatrix} 0 \\ g(m_1+m_2) \\ g(m_1r_1+m_2l_1)s_1+gm_2r_2s_{12} \\ gm_2r_2s_{12} \end{pmatrix}$$

where $s_1=\sin(\theta_1)$, $c_1=\cos(\theta_1)$, $c_{12}=\cos(\theta_1+\theta_2)$, etc.

The external force vector, *F* allows us to apply friction and motor torque to the system. Motor torque, if any, is inserted into the third row of the *F* vector, corresponding to a force applied to the relative θ_2 angle between the joints.

These equations describe the 2-link system with four degrees of freedom. Numerical integration of these general equation *(1) with the given matrices will result in a simulated MonkeyBot in unrestricted free-flight motion.

3.2 Reduced Dynamics for Swinging State

While the free-flight dynamics fully describe the unrestricted system, we need to contstrain one of the link ends in order to take advantage of the MonkeyBot's swinging properties. In the swinging state, the robot is assumed to rotate around the fixed end at *(x,y)*. Instead of applying constraints to the equations of 3.1, the system can be reduced to only two generalized coordinates, $q=[\theta_1\ \theta_2]^T$. Solving this reduced system follows the procedure above (though the algebra is much simpler). The resultant matrices for the standardized form *(1) in this swinging state are:

$$q=\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}$$

$M(q) =$

$$\begin{pmatrix} I_1 + I_2 + m_1 r_1^2 + m_2 l_1^2 + m_2 r_2^2 + 2 m_2 r_2 l_1 c_2 & m_2 r_2^2 + m_2 r_2 l_1 c_2 + I_2 \\ m_2 r_2^2 + m_2 r_2 l_1 c_2 + I_2 & I_2 + m_2 r_2^2 \end{pmatrix}$$

$C(q, \dot{q}) =$

$$\begin{pmatrix} -m_2 r_2 l_1 s_2 \dot{\theta}_2 & -m_2 r_2 l_1 s_2 (\dot{\theta}_1 + \dot{\theta}_2) \\ m_2 r_2 l_1 s_2 \dot{\theta}_1 & 0 \end{pmatrix}$$

$G(q) =$

$$\begin{pmatrix} g(m_1 r_1 + m_2 l_1) s_1 + g m_2 r_2 s_{12} \\ g m_2 r_2 s_{12} \end{pmatrix}$$

This swinging state is applicable to any brachiating type motion. The robot might swing with only one magnet attached to the wall - a situation that is described by these reduced equations. At the end of the swing, the robot could release the magnet and enter a free-flight phase, which would instead be described by the full 4 degree of freedom system in section 3.1 Smooth transitions between these states is critical to an accurate simulation.

3.3 Impact Dynamics

To move any significant distance across the wall, the MonkeyBot needs to transition its holding points by changing which magnet is fixed. The transitions involve impact and corresponding loss of energy as the magnet attaches to the wall surface. It is assumed the friction and holding force of the magnets is high, so we model these collisions as purely plastic.

If both magnets are simultaneously active, then the solution to this impact is trivial and uninteresting – all motion and velocities are halted. As such, we will only consider impacts in the case of transition from free-flight to swinging dynamics (or, equivalently, an instantaneous change from one magnet to the other).

Using the end of the top link as the attachment point, we approximate the impact as an (unknown) finite impulse applied at the *(x,y)* position. The impulse is constrained so that the fixed end velocities in the post-impact state are zero (plastic collision).[ii]

$$J(q)\dot{q}^{+} = 0 \qquad (*)$$

J(q) is a Jacobian matrix that transforms the generalized velocities (*dq*) into a secondary coordinate system consisting only of the velocities we want to become zero: [*dx dy*]^T. *dx* and *dy* are identical in both systems, so *J(q)* is simply:

$$J(q) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Impulse applied to a system causes a change in momentum (mass*velocity). The change in momentum is reflected by a change in velocity, Δdq=(dq^+ - dq^-), where dq^- and dq^+ are the pre- and post- impact velocities respectively.

$$M(q)\Delta\dot{q} = J(q)^T\lambda \quad (**)$$

The parameter λ in (**) represents the (unknown) impulse applied to the fixed end during.

Combining equations (*) and (**), and solving to eliminate the parameter λ, gives us the solution to our impact problem. The post-impact velocities are in terms of the pre-impact state and a projection matrix *P(q):*

$$\begin{aligned} \dot{q}^{+} &= P(q)\dot{q}^{-} \\ P(q) &= I - M^{-1}J^T(JM^{-1}J^T)^{-1}J \end{aligned} \qquad \textbf{*(2)}$$

The above equation applies when the end at (x,y) becomes fixed. In the alternate case where we desire to activate the other magent (at the end of the second/bottom link), we can simply transform the coordinates so that *(x,y)* are located about the desired point. Applying geometry to *q* and *dq*, the bottom and top links are effectively interchanged. These transformations are:

$$q_{flipped} = \begin{bmatrix} x \\ y \\ \theta_1 \\ \theta_2 \end{bmatrix}_{flipped} = \begin{bmatrix} x + L_1\sin(\theta_1) + L_2\sin(\theta_1+\theta_2) \\ y - L_1\cos(\theta_1) - L_2\cos(\theta_1+\theta_2) \\ \pi + \theta_1 + \theta_2 \\ -\theta_2 \end{bmatrix}$$

$$\dot{q}_{flipped} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}_{flipped} = \begin{bmatrix} \dot{x} + L_1\cos(\theta_1)\dot{\theta}_1 + L_2\cos(\theta_1+\theta_2)(\dot{\theta}_1+\dot{\theta}_2) \\ \dot{y} + L_1\sin(\theta_1)\dot{\theta}_1 + L_2\sin(\theta_1+\theta_2)(\dot{\theta}_1+\dot{\theta}_2) \\ \dot{\theta}_1+\dot{\theta}_2 \\ -\dot{\theta}_2 \end{bmatrix}$$

We must also ensure to swap the parameters for the links, to ensure the “flipped” system is consistent with the old one. The impact equations ***(2)** can then be correctly applied to dq_{flipped}.

4. Attempts to Find Stable Open-Loop Gaits

4.1 Horizontal Brachiating Gaits

Using the simulation, we can attempt to identify stable gaits for use by open-loop controllers.

The simplest motion to target for the stable gait search is horizontal brachiation. This involves swinging movements with end of one link fixed to wall at all times. We can start the search for a specific subset of gaits that contain a single impact, are symmetric about the vertical position, and go through the following steps as in *Figure 2*:

1. Initial position at vertical (θ_1=0, θ_2=0) with a set of initial velocities $d\theta_1$, $d\theta_2$. The robot has one link (labeled ‘A’) fixed to the wall.
2. Initial velocities cause the end of second link (‘B’) to swing up to the same vertical height as A.
3. The magnet at B fixes to the wall as A is released (an instantaneous switch). This causes an impact with corresponding change in velocities.
4. Point A begins to swing downwards. Torque is applied (from the motor) to the central joint until energy is fully restored to the system.
5. Gait ends when position passes through vertical (θ_1=0).

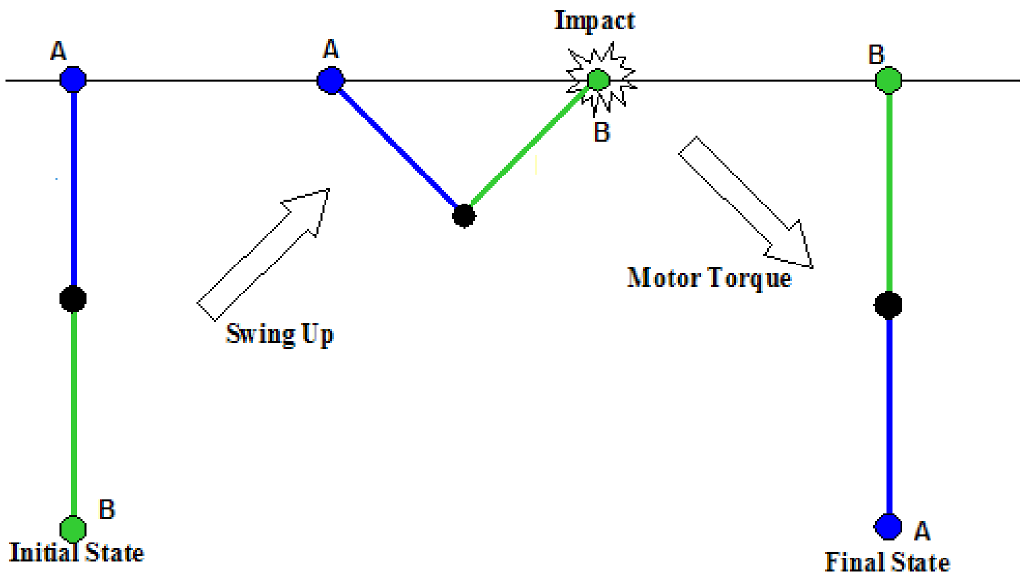

Figure 2: Stages of robot movement during the symmetrical horizontal brachiating gait.

If the final state and initial state (θ_1, θ_2, $d\theta_1$, and $d\theta_2$) are identical, then the gait is repeatable and causes only purely horizontal movement. Continuing the gait continually will allow the MonkeyBot to move sideways indefinitely without changing height.

4.2 Searching for Stable Gaits

A search algorithm was employed to identify these horizontal brachiating gaits, and subsequently determine if they are stable. The search parameters can be defined concisely by the set of two initial conditions ($d\theta_1$, $d\theta_2$). The system starts with these initial conditions and is then integrated until point B reaches vertical (checked by the condition $\theta_2 = \pi - \theta_1$). At this time the magnets are switched, impact projections applied, and motor turned on. As point 'A' descends, the motor is turned off when the energy is restored to its initial state value. The integration is finally halted as the top link passes through vertical (θ_1=0). When this condition occurs, an error function for closeness to the initial state is evaluated. The error is defined as

$$\left(\theta_{2\_Final}\right)^2 + \left(\dot{\theta}_{1\_Initial} - \dot{\theta}_{1\_Final}\right)^2 + \left(\dot{\theta}_{2\_Initial} - \dot{\theta}_{2\_Finall}\right)^2$$

Minimizing the error allows us to identify the horizontal brachiating gaits. If a successful gait is identified, recording the times of magnet switching and applied motor torque allows us to produce it with an open-loop controller.

An initial search was iterated over a large range of values to gain insight into the nature of the search space[iii]. Results of this iteration are displayed as a 3D plot in *Figure 3* and as an intensity map in *Figure 4.*

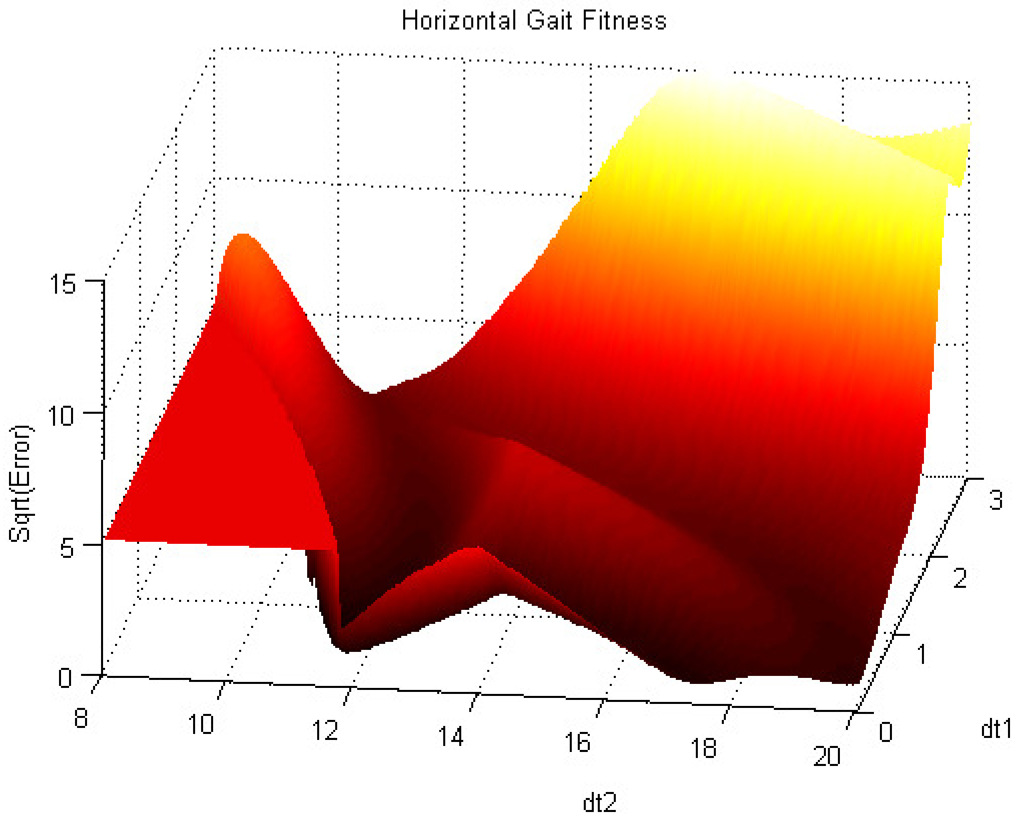

Figure 3: 3D Plot of error value square root for range of initial conditions $d\theta_1$, $d\theta_2$. Flat area in bottom left are sets of initial conditions without sufficient energy to bring point 'B' to horizontal.

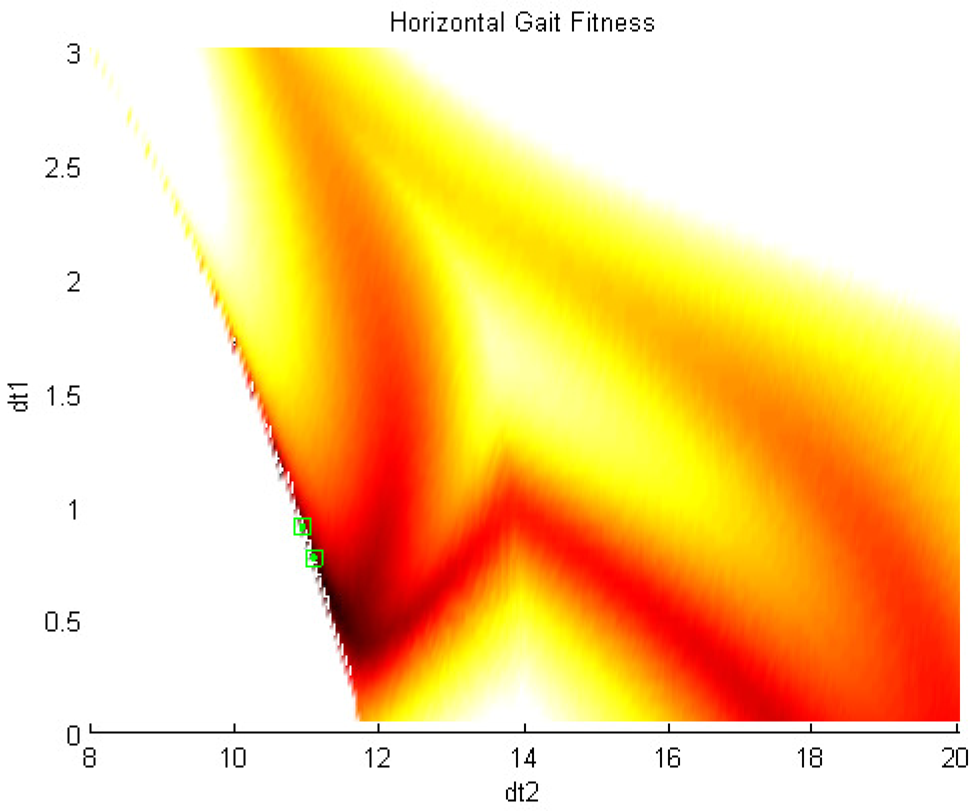

Figure 4: Intensity map of error function values. Error values above 5 have been truncated to increase the utility of the graph. Darker colors (black) indicate smaller values for error function. Green squares/dots indicate locations of indentified minima.

Darker values indicate a smaller value of the error function. The area of smallest values can be seen around ($d\theta_1$, $d\theta_2$) = (0.5, 11.5). Interestingly, this is along the border of the area where initial conditions did not have sufficient energy to bring point 'B' up to vertical. This seems to indicate that minimizing energy gives the best values for our error functions.

A finer tuned optimization, with initial guess of (1, 12) identified the minimum of .000016 at (0.902, 10.935). The corresponding event times were magnet switch at 0.428 seconds with 0.025 seconds of motor torque.

A preliminary evaluation of stability is to run the simulation for several cycles. Although initially performing well, the repeatability of the gait failed after running for several iterations, as shown in *Figures 5-7*. The error is likely due to the limited precision of 1ms used for the magnet/motor controllers in the simulation. Desired stable gaits, for use in motion planning, would continue to travel horizontally despite this and even be able to correct small changes in initial conditions.

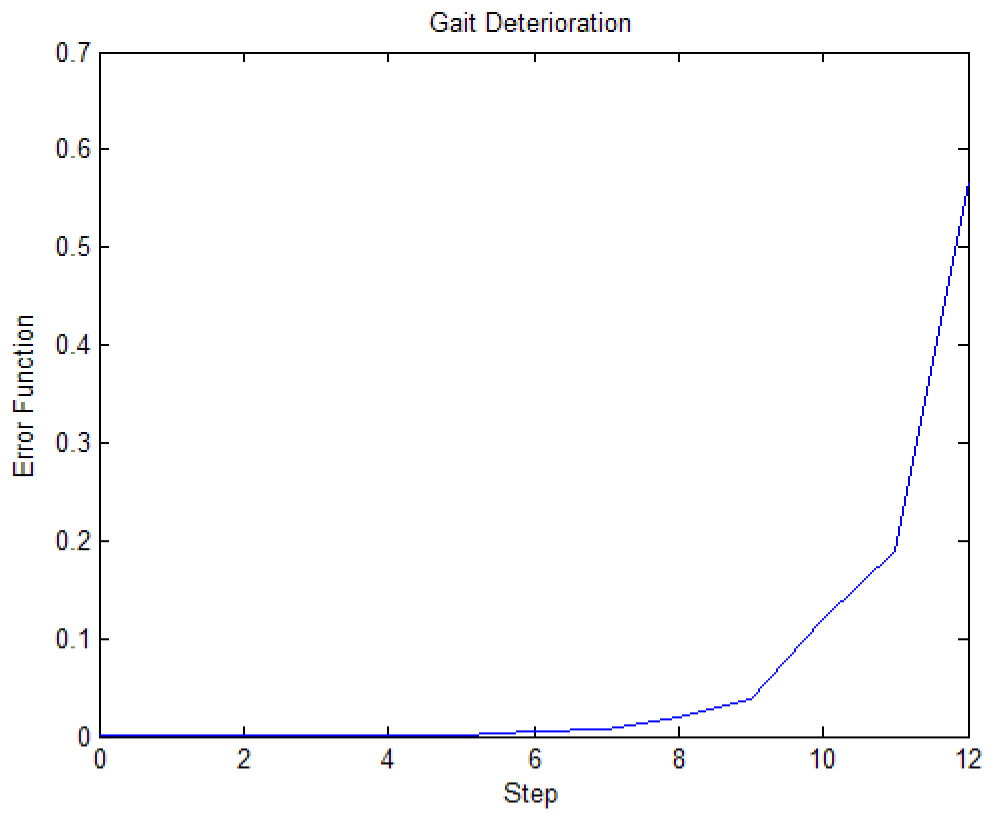

Figure 5: Error value (difference from initial conditions when θ_1=0) while iterating the first discovered gait (0.902, 10.935).

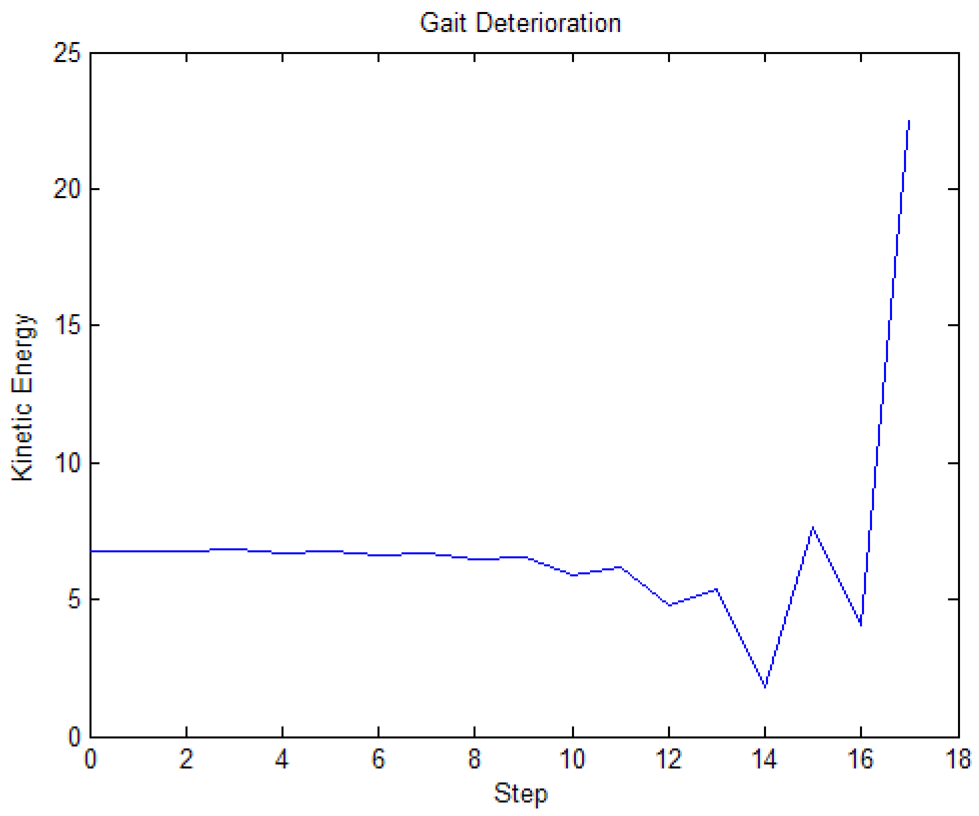

Figure 6: Kinetic energy of robot at vertical position (θ_1=0) during each iteration. Kinetic energy starts to change visibly around gait iteration 10, correlating with the increase in error value.

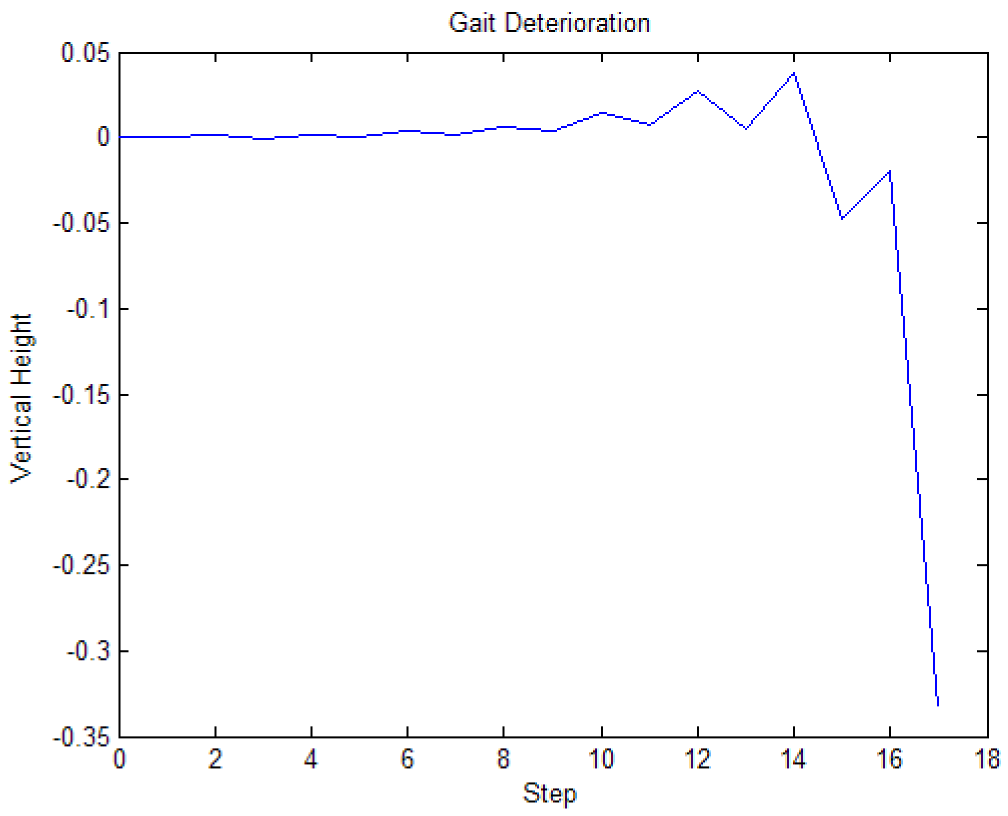

Figure 7: Vertical height (at impact point) while repeating the first discovered gait (0.902, 10.935). The robot maintains the same approximate vertical position during initial iterations, but begins to fall as the error increases. Each link is 0.61m in length.

A second search, starting with a guess of (.5, 11.75), identified another minimum of .00053 at (.768, 11.094). However, running this gait for several iterations gave similarly undesirable results. *Figure 8* shows a plot of the error function for this gait.

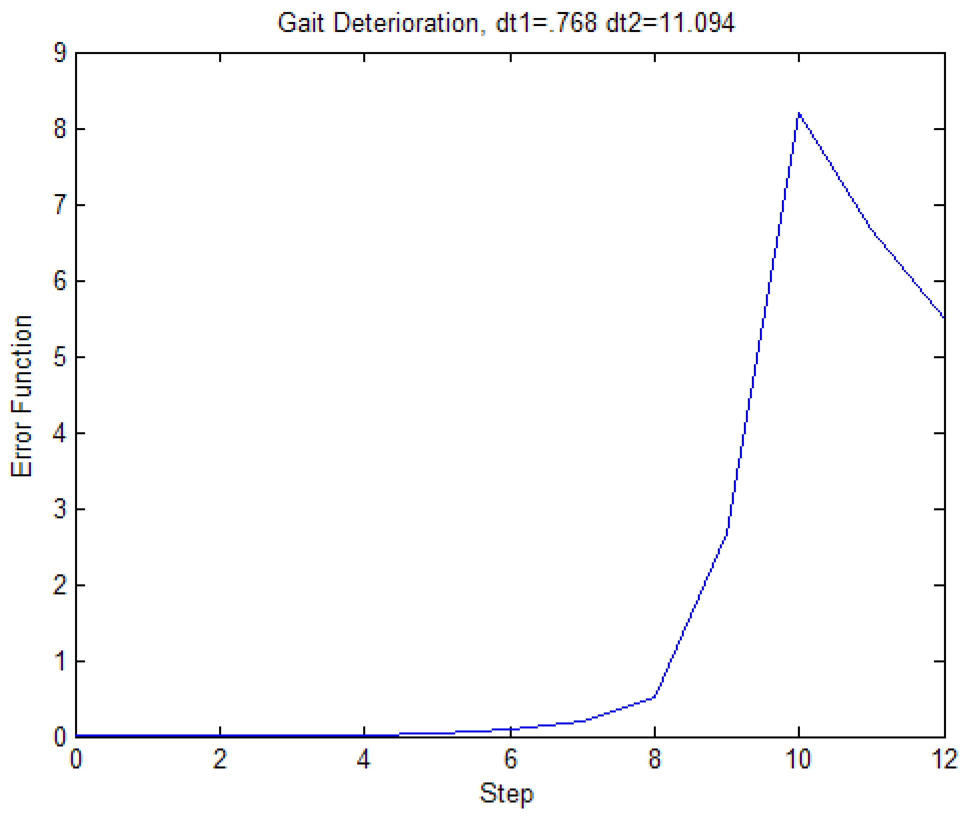

Figure 8: Error value while attempting second gait (0.768, 11.094).

Additional graphs:

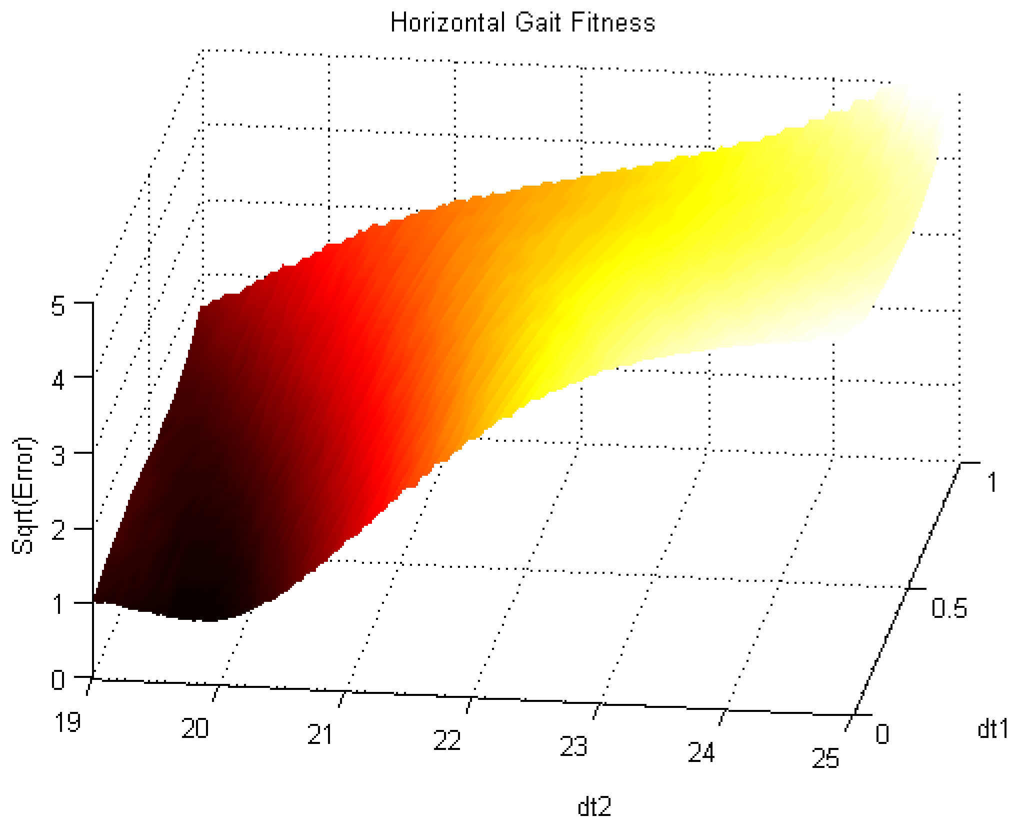

Figure 9: Error values for higher initial $d\theta_2$. The minimum of this graph, near $d\theta_2$=20, does not drop much below an error value of 1. Values above 5 truncated.

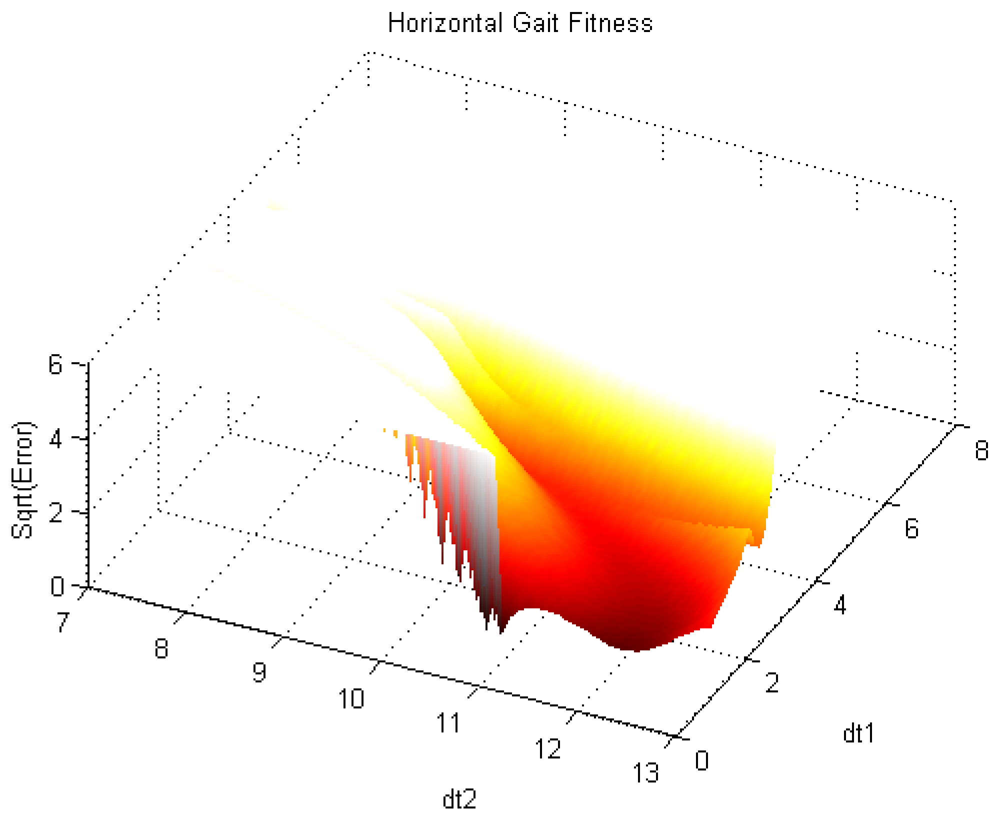

Figure 10: Error values for lower values of dθ2 and higher values of dθ1. Same results as *Figure 9*.

4.3 Future Work

These preliminary results indicate there are no stable gaits belonging to the subset defined in section 5.1. Expansion on this research could look at variations of the motions tested, including gaits that:

- Have more than one impact (i.e. multiple swings per iteration)
- Involve free-flight, with both magnets released during part of the travel
- Applying the motor at different or multiple times
- Do not pass through and are not symmetrical about the vertical (θ_1=θ_2=0)

Testing any of the above modifications may result in a more stable gait that is suitable for motion planning. Changing the system parameters including mass, link lengths, etc. may also benefit the search.

Additionally, there may be stable solutions for other motions that are not purely horizontal, such as climbing gaits.

5. Simulation

5.1 Code High-Level Overview

The primary objective of the code is to provide a simulation for MonkeyBot. The code, written in Matlab, handles equations of motions for all possible motion types: free-flight, swinging, and fixed. It also smoothly transitions between motion types, modeling impacts as necessary. Parameters for the MonkeyBot are user modifiable.

The simulation function, *monkeySim* is called with the initial conditions and system parameters. It will return results of the simulation at each time step (.001seconds). Numerical integration is done with Matlab's *ode45*, using the standardized equation ***(1)** and the matrices applicable to the motion state / magnet configuration.

Solving the equations of motion for each time step is handled by the function *DiffEqWrapper*. The control algorithm is (mostly) modular and contained with *controlAlgorithm.m*. It can be easily replaced or modified.

The attached .zip file contains a folder with the simulation code, and a sample script file *main.m* to demonstrate its use.

The overall algorithm is as follows:

1. Setup script: *main.m* – sets up the initial conditions, system parameters, and controller parameters
2. Simulation function: *monkeySim* is then called with all necessary parameters
3. Values for generating the matrices *M, C,* and *G* are pre-calculated to improve speed
4. *ode45* is called to numerically integrate the equations of motion
5. For each time step in ode45, the equations of motion are solved by *DiffEqWrapper*
    a. Reduced matrices (2 DOF) used when swinging with only one magnet attached
    b. Full matrices (4 DOF) used when neither magnet is attached.
6. For each time step specified (.001seconds), a monitoring function, *DiffEqMonitor* is called by ode45
    a. Plots the location and state of the MonkeyBot, so that user can watch simulation progress
    b. Control algorithm is called to check current state
        i. Control algorithm receives current system state Q
        ii. Can adjust motor torque or change magnet states
7. Changes in motor / magnet state restart the integration so that discontinuties are handled correctly
8. If the magnets are transitioned (impact):
    a. System state is flipped (bottom link becomes top, end point becomes new base x/y)

    b. Projection matrix calculates post impact velocities from free-flight dynamics.

9. Integration repeated with new initial conditions

5.2 Adjusting Control Algorithm and System Parameters

System parameters available for modification include *m, I, L, and r* for each link, as well as the gravity constant *g*. These parameters are passed in vector form (e.g. M=[m_1 m_2]) to the simulation function. Demonstration of these parameters is shown in *main.m.*

The control algorithm setup was designed to be isolated from the rest of the simulation. The function *controlAlgorithm* is called for each time step, and is passed the current time and system state. Any changes made to the global control variables (motorState, magnetState) will be reflected as the simulation progresses.

The current code package includes a simple time-based (open-loop) controller. Modification of initial_t, motor_t, swap_t, and maxMotorPower will give demonstration of varying simulation results. See *main.m* and *controlAlgorithm.m.*

Additional parameters can be fine tuned in various parts of the program:

- Step size (currently .001) can be changed in *monkeySim.m* to change the controller resolution or number of reported values. *Note:* This does not affect the accuracy of the integration as *ode45* chooses its own step size then interpolates to return values at the desired points. It does affect the precision of when the controller acts.
- Friction (currently 0). *DiffEqWrapper* includes parameters to apply both constant and viscous friction to the joints at θ_1 and θ_2.

Frequency of graphing can be changed in *DiffEqMonitor.* This has a large impact on simulation time.

5.3 Searching for Stable Gaits Using Simulation

See *GaitSearchQuickStart.pdf* embedded within code package.

Quick Start:

The search for gaits involves modified versions of the simulation functions designed to search through range of parameters.

3D Plot for Error Values

The ***search.m*** script file contains everything necessary to iterate over a range of values and create 3D plots of the fitness function.

Modify the values for the range/arrays *dt1_rng* and *dt2_rng*. The script will iterate over all combinations of values contained within this range, and store the error values in the *results* array. Everything is saved to a time-stamped *.mat* file, and the square root of the values is plotted automatically.

Minimization/Optimization of Initial Values

The ***minimize.m*** script file contains the procedure necessary for finding and evaluating gaits.

Modify the seed values for the minimization: *dt1_guess*, *dt2_guess*. The script will use the Matlab function *fminsearch* to find the optimal solution. This solution is then iterated (and shown visually) for several cycles. Results for error value, kinetic energy, and, vertical height, are plotted.

Gait-Search Specific Code

The main component in the above two scripts is a modified version of the simulation function, called *runSim_fitness*. This function runs the simulation only through the steps of a single gait (swing up, impact, motor impulse, finish at vertical). At the end of the single gait, it returns the fitness value calculated as the sum of the squares of the difference from initial conditions. *runSim_fitness* is evaluated for every point in the 3D plot, and minimized in the optimization step.

The fitness function uses a modified version of the controller, *controlAlgorithm_search*, that acts depending on the state of the robot. It swaps the magnets at when the second link has reached the horizontal ($\theta_2=\pi-2\theta_1$) , and impulses the motor until pre-impact energy matches post-impact energy. The timings of these events are recorded in global variables, which are obtained later by the script file as necessary.

A second modified simulation function, *runSim_stability* runs the robot through several iterations using the standard open-loop controller. It calculates some potentially useful values at each iteration.

Modifying the Search / Looking for different types of Gaits

To modify the search or looking for different types of gaits involves three main steps:

1. Change the search controller, *controlAlgorithm_search*, to act and record timings as necessary for the gait.
2. Change the fitness simulator, *runSim_fitness*, to stop integration at the right point, and return the correct fitness value.
3. Change the actual control algorithm, *controlAlgorithm*, to reflect the modified gait type.
4. Modify the script files *search.m, minimize.m* as necessary to use new gait parameters / controller timings.

---

[i] *M(q), C(q,dq), G(q)* matrices for both dynamic systems (free-flight/4 DOF and swinging/2 DOF) obtained from Nelson Rosa.

[ii] Equations for impact dynamics obtained from Kevin Lynch.

[iii] System paramters (Mass, Inertia, etc.) and starting point for parameters dθ1 and dθ2 based off of 2-link model in *"A five-link 2D brachiating ape model with life-like zero-energy-cost motions"* by Mario W. Gomes & Andy L. Ruina.